The Sting Racing Team's Entry to the Urban Challenge

Matthew Powers, Dave Wooden, Magnus Egerstedt, Henrik Christensen, and Tucker Balch

1 Introduction

Every mobile robotics project requires that several basic problems be addressed. The robot must sense the world and react to it. This requires reliable and effective computing, sensing, and locomotion hardware as well as robust software. On top of this, the Urban Challenge presented a new and unique problem in the space of the active robotics programs: the presence of multiple independent autonomous agents in a semi-structured but diverse and complex environment. We can contrast this with other contemporaneous DARPA projects, (*e.g.* UPI/Crusher, the LAGR project [**?**, **?**]) in which the world is static and the robot itself is the primary moving object.

To model the dynamic complexities of the environment in the Urban Challenge, we choose to represent the various situations the robot can be placed in using a nested hybrid automaton, where each state corresponds to a particular scenario, defined at different levels of abstraction. This approach leads to an elegant decomposition of a very large problem into manageable subproblems. Above the automaton

Matthew Powers
College of Computing, Georgia Institute of Technology, Atlanta, GA 30332
e-mail: mpowers@cc.gatech.edu

Dave Wooden
Boston Dynamics, Waltham, MA 02451
e-mail: dwooden@gmail.com

Magnus Egerstedt
Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332
e-mail: magnus.egerstedt@ece.gatech.edu

Henrik Christensen
College of Computing, Georgia Institute of Technology, Atlanta, GA 30332
e-mail: hic@cc.gatech.edu

Tucker Balch
College of Computing, Georgia Institute of Technology, Atlanta, GA 30332
e-mail: tucker@cc.gatech.edu

lies a mission planner, directing the robot to its next major objective. Below the automaton lies a control system composed of reliable parameterized control primitives responsible for the robot's actuation. Externally, perception processes provide fused sensing information based on a wide array of sensing modalities.

The Sting Racing Team was composed of engineers from Georgia Tech, the Georgia Tech Research Institute, and Science Applications International Corporation (SAIC). Our base vehicle was a Porsche Cayenne (shown in Figure 1) and modified to support a bay of COTS computers, LIDAR and RADAR scanners, cameras, GPS, and an IMU.

Fig. 1 The Sting Racing retrofitted Porsche Cayenne entry to the DARPA Urban Challenge.

This chapter is broken into three main sections. In Section 2, we describe the hardware components of our vehicle, including the base automobile, the sensors installed, and the computing resources. In Section 3, we describe how the world is sensed and how actions are planned, with special attention given to our approach for hierarchically decomposing the various situations which the vehicle must be able to handle. In Section 4, we discuss testing and performance of our robot at several different sites including at the DARPA National Qualifying event in Victorville, California.

2 Hardware Architecture

Our approach to hardware design for the Urban Challenge was driven by a number of considerations. These considerations include the DARPA-defined design requirements, the requirements of available software approaches, and careful consideration of the demands of the task and environment. A discussion of the team's hardware approach and the factors driving the most important decisions follows.

2.1 Base Platform

Our base vehicle platform is a 2006 Porsche Cayenne modified to be controlled via on-board computers. An AEVIT driving control system integrated by Electronic Mobility Controls (EMC) provides primary servo control of steering and acceleration/braking as well as controls for secondary functions, including ignition, transmission, lights, and the parking brake. The drive-by-wire system allows for easy switching between autonomous and manual operation. Additionally, the hardware configuration allows 4 people to ride in the vehicle while in autonomous mode, which is a major convenience during evaluation and testing.

The computing system was comprised of 8 Dual-Core Intel XEON 5120 processor-based computers, connected by dual gigabit ethernet networks. This computing power provides the resources to process the perception of the vehicle and the environment, control the vehicle, and log data. Power for the AEVIT servo and vehicle system controls is provided by the Cayenne 12V DC system. All sensors and associated computer equipment are powered from an auxiliary, engine-driven, 24V DC alternator configured to provide 75A at idle. This exceeds the sensor suite and computer equipment power requirements of 50-55A, 24V DC during operation and provides additional power for future requirements. Batteries, contractors, circuit breakers, and the power distribution panel for the 24V system are located in the rear vehicle compartment with the computer rack. DC-DC converters are installed to provide power for sensor and peripheral equipment requiring 12V or 5VDC.

Roof-mounted equipment includes an amber safety strobe and emergency stop pushbuttons accessible from either side. The safety strobe and an audible warning signal provide an indication of autonomous operation. Antennas are installed for remote pause and disable receivers, and an 802.11G WLAN interface is provided for developmental purposes. Other safety-related equipment includes internal pause and emergency stop controls and the circuitry needed to disable the vehicle, power plant, sensor suite, and computer equipment when commanded.

2.2 Sensors

2.2.1 LIDAR

Our team installed four flavors of LIDAR scanners on the robot. The primary obstacle detection sensors on the system are the SICK LD-LRS-1000 planar scanners, two of which are mounted on either side of the front of the vehicle (see Figure 2(a)). These sensors have an angular resolution of 0.5°, effective range of about 100m, and a spin frequency of 10 Hz. These two sensors cover nearly 360° field of view around the robot, with the exception of the region behind the robot's back bumper, which is covered by a different sensor.

In addition, three other planar LIDAR scanners are mounted on the front bumper. Two SICK LMS-291 scanners are mounted next to the LD-LRS-1000s and have a

Fig. 2 (a) One of the two SICK LDLRS-1000 LIDAR scanners (blue) and one of the seven SICK LMS-291 LIDAR scanners (beige). (b) The Riegl LMS-Q120 LIDAR scanner. (c) The four roof-mounted SICK LMS-291 units, mounted for use as in a "pushbroom" manner. (d) Rear-mounted SICK LMS-291.

scan rate of 75 Hz, field of view of 180°, effective range of 40 meters, and angular resolution of 1/4°. Mounted on the center of the front bumper is a Riegl planar scanner, with a field of view of 80°, angular resolution of 0.25°, and range of more than 120 meters. These three scanners provide redundant coverage to the scanning area of the LD-LRS-1000. Another SICK LMS-291 is mounted on the center of the back bumper, providing coverage of the rear of the robot.

Four SICK LMS-291 scanners are mounted on the roof in a "pushbroom" manner (*e.g* see [**?**]). These provide detection of geometric features of the road like curbs, and are also used to detect vehicles in front of the robot. We also attempted to use these to detect road markings based on reflectivity information, but the signal to noise ratio was not sufficient for this to be a reliable source of information.

2.2.2 Radar

For collision warning radar, we selected an Eaton Vorad EVT-300 as a means of providing reliable detection for oncoming vehicles at large distances. The EVT-300 emits 3mW of RF power at 24.725 GHz, with 1 MHz bandwidth. Able to detect and track as many as 20 objects at up to 150 m, it can update seven tracks at 15 Hz. A motorcycle-sized target can be detected at distances as great as 110 m. Target range (+/- 3 feet), velocity, and azimuth are included for each tracked target. With approx-

Fig. 3 The Eaton Vorad RADAR units.

imately a 12° field of view, multiple EVT-300 units are required to provide adequate warning for the front and sides of the vehicle. One front-mounted unit provides redundancy for the Riegl, as well as the complementary capability to maintain tracks independently of other system software. Four side-mounted units are placed near the opposite ends of the front bumper to provide indication of approaching traffic as the vehicle pulls into an intersection or out of a side road. Figure 4 shows the sensor configuration mounted on the vehicle.

2.2.3 Cameras

Six Prosilica GC-650 Gigabit Ethernet color cameras provided images to the front, sides and rear of the vehicle. They are capable of providing 12 bits of intensity dynamic range per pixel, and the fast Ethernet interface supports high frame rates. All 6 cameras were roof-mounted in environmentally sealed enclosures. The two forward-looking cameras are used for road following.

2.2.4 GPS and Inertial Sensors

The vehicle was equipped with the Novatel SPAN inertial navigation system, consisting of the Novatel Propak G2plus GPS receiver with roof-mounted antenna and a Honeywell HG1700 AG17 inertial measurement unit mounted over the rear differential, inside the vehicle. The GPS unit receives both L1 and L2 signals and is capable of pseudorange differential corrections. Omnistar High Precision corrections are used to improve GPS accuracy. The IMU allows the vehicle's position to

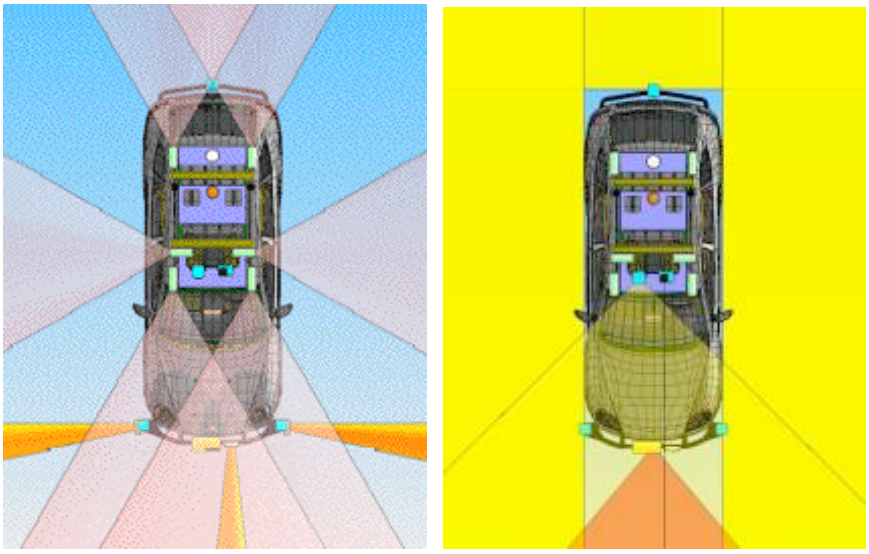


Fig. 4 In (a), sensor placement and coverage for six Procsillica GC 650 color Gigabit Ethernet cameras (pink) and three EVT-300 radars (orange). In (b), sensor placement and coverage for five SICK LMS 291 LIDARs (yellow) and one forward-facing Riegl LMS-Q120 LIDAR (orange).

Fig. 5 (a) One of the six roof-mounted Prosilica cameras. The forward-mounted cameras were used as the primary lane-sensing tool. (b) The GPS antenna, mounted on the roof of the vehicle.

be estimated during periods of GPS dropout, as are expected in an environment with limited view of the sky. The HG1700 IMU specifications are given in Table 1. The SPAN technology described by Novatel [**?**] combine WAAS-corrected GPS readings with IMU data to provide position accuracy with 0.8 m CEP, velocity accuracy of 0.02 m/s RMS (nominal), attitude accuracy of 0.015° (pitch or roll) and 0.05° (yaw), and acceleration accuracy of 0.03 m/s2. In referenced tests, an error of 1-3m was seen during a 60s dropout period, depending on how many GPS satellites were blocked. Additionally, the time to reacquire a GPS position after a dropout was improved dramatically, from 11s to 1s, by keeping an estimated position based on IMU data.

Gyro Input Range	± 1000 degrees/sec
Gyro Rate Bias	10.0 degrees/hr
Gyro Rage Scale Factor	150 ppm
Accelerometer Range	± 50 g
Accelerometer Linearity	500 ppm
Accelerometer Scale Factor	300 ppm
Accelerometer Bias	3.0 mg

Table 1 IMU specifications for SPAN INS.

2.3 Computing Resources

Fig. 6 The rack of eight computers mounted in the trunk of the Cayenne.

A rack of eight computers was mounted in the trunk (Figure 6). Each computer was equipped with a dual-core Intel XEON 5120 processor and 4GB of RAM. A Gigabit Ethernet network provided connectivity between all the computers, in addition to providing an interface to the six roof-mounted cameras and the four bumper-mounted RADAR units. The eight computers were utilized in the following way:

- Reactive steering and speed control (one computer)
- Mission, route, and situational planning (one computer)
- LIDAR/RADAR processing (two computers)
- Static and dynamic obstacle detection and mapping (one computer)
- Lane tracking (one computer)
- Data logging (one computer)
- Spare (one computer)

Communication between processes on one computer was implemented using shared memory, while communication between processes on different computers was implemented over the Ethernet network. Highlights of functionality of these software processes are discussed in detail in Section 3.

3 Technical Approach

Our overall approach was motivated by the ambition to drive on road networks with sparse GPS waypoints (one marker per 20 meters) with poor GPS reception. This necessitated driving based on local perception (LIDAR and camera based sensing) rather than driving based on detailed prior knowledge of the road or precise global positioning of the vehicle. Generally speaking, our aim was to produce a system that could find its own way while driving down road segments, negotiate intersections as it came to them, and deal with unexpected events along the way.

A final concern central to our design was the notion that the robot must strictly obey the rules regarding vehicle spacing and separation as specified by DARPA. This meant that the robot would not drive next to another vehicle or any other obstruction that brought it closer than 1 meter on the sides or several meters in front (depending on vehicle speed).

3.1 High-Level Architecture

From a high-level standpoint, the software architecture follows a hierarchical, hybrid approach. Notable is the combination of a sequential, deliberative and reactive layers, combined in a manner similar to classical three-tiered systems [**?**, **?**], as shown in Figure 7. The mission-level strategic planner provides a symbolic plan over the high-level graph-based map, recommending a mode of operation (such as drive along paved road, drive along unpaved road, drive through intersection, or drive in unstructured area) along each edge in the graph. This plan is passed on to the tactical deliberative layer and the sequential layer in the form of a commanded task.

The tactical deliberative layer maps the commanded task to a spatial planning goal. For some tasks (e.g. “drive to a point in an unstructured environment”), the goal is a location in the global map. For other tasks (e.g. “follow paved road”), the goal is to navigate along a perception-based signal (e.g. the detected road) as far as the horizon of usable perception.

The sequential layer builds upon the paradigm of the finite state automata, implementing a nested hybrid automaton [**?**]. Nested hybrid automata extend the functionality of hybrid automata, and are explained in Section 3.6. Input from the strategic deliberative layer is interpreted as discrete input to the transitions in the top level automata.

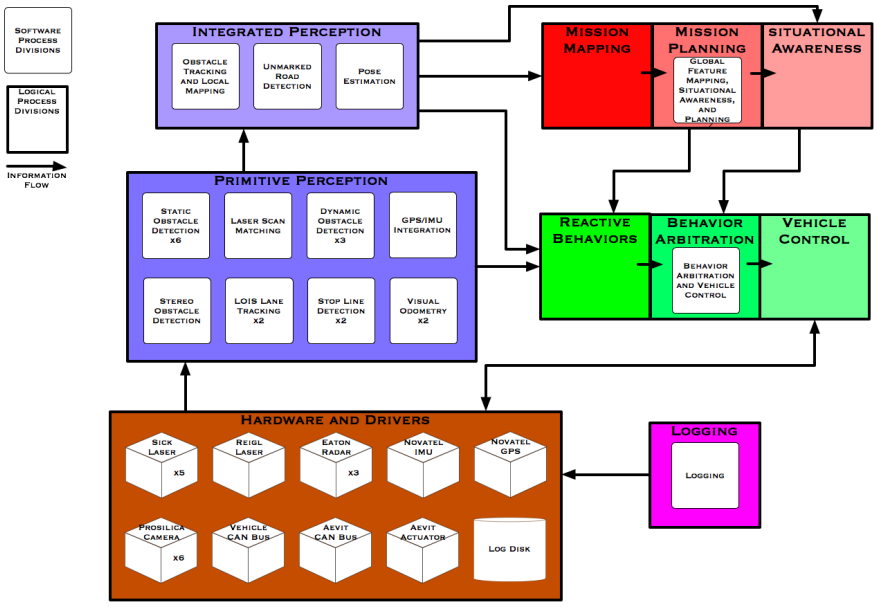


Fig. 7 Software processes used within the Sting software architecture and their relationship to the conceptual architecture presented in Figure 1. Smaller boxes represent divisions of labor between software processes (e.g., Static Obstacle Detection). Larger boxes represent divisions of labor within the conceptual architecture (*e.g.*, Primitive Perception).

The reactive layer is implemented by a voting scheme over circular arc paths at increments of curvature from the robot's current position. The state of the sequential layer is mapped to a weighting scheme over the voting behaviors, and the plan produced by the tactical deliberative layer is interpreted as a series of goal locations by one or more of the behaviors.

3.2 Static and Dynamic Obstacle Detection

The robot is equipped with three types of obstacle sensors: horizontal plane LIDAR scanners, pushbroom LIDAR scanners, and RADAR scanners. Each sensing modality is processed separately to detect relevant features. The features are then fused to create a coherent representation of the robot's surroundings.

A point cloud is collected from the horizontal LIDAR scanners at regular intervals. This cloud is segmented into a list of distinct objects based on spatial spacing. The objects in each such list is tracked from iteration to iteration based on a greedy matching algorithm. A first-order heuristic-based estimator computes a world-frame velocity for each object, based on its position change over time.

The downward facing pushbroom LIDAR data passes through a filter to extract relevant features such as car profiles and curbs from the ground data. These features are incorporated with the horizontal plane LIDAR data at the initial object detection level. The Eaton Vorad RADAR devices return position and velocity estimates for up to twenty independently tracked objects, per device. This is combined with tracked object information to supplement the object list and improve the velocity estimate generated by the LIDAR data.

As a final step, each object is identified to be static or dynamic, based on its velocity estimate over its lifetime. In addition, in order to support various components of the planning and control modules, we use the geometric and velocity information of each object to assign a classification. The classification comes from the set $\{$ *car, wall, ground, other* $\}$. For example, the *car* class of objects are used for establishing precedence at an intersection.

3.3 Lane and Road Detection

The primary behavior of the robot is to drive down a road. In order to do so, the robot must know its location relative to the road. GPS positioning is not accurate enough to perform this function, even with Omnistar High Precision corrections. Furthermore, the GPS lane information and satellite imagery supplied by DARPA could not necessarily be relied upon to be as precise as needed.

We pursued an approach that estimated the relative position and curvature of the lane based on visual clues from monovision cameras [**?**, **?**, **?**]. This approach is capable of detecting not only the white and double yellow lane markings, but also could detect the apparent luminance deviation that occurs on a curb, as demonstrated in Figure 8. This approach has the benefit of being completely independent of any external information source (such as a Route Network Definition File (RNDF) or *a priori* satellite imagery).

Fig. 8 Two examples of the paved-road lane tracking used on the Sting vehicle. The detected lane is indicated by red dots.

Because the vision system could be confused or washed out, often without cognizant failure, we used a statistical combination of the GPS lane information with the visual lane perception to form a fused lane estimate. Essentially, when the visually perceived lane was within a corridor surrounding the GPS lane estimate, the vision-based lane was given greater weight. When the lane estimate deviated, more weight was given to the lane estimate based on GPS and clues from detected curbs. In addition, the speed setpoint of the robot was reduced.

3.4 Intersection and Vehicle Detection

One of the key perception challenges of the Urban Challenge is to determine when cars are present or approaching an intersection. This requires that the robot be able to detect stationary and moving cars, estimate their position relative to the intersection, and properly ignore other detected environmental features and clutter.

The robot knows that it is approaching an intersection based on its distance to specially-marked node in the RNDF graph (see Section 3.5). As it comes to within 15 meters of the stop line, it enables the vision-based stop line detector for precise alignment with the intersection, since the GPS information is not reliable or accurate enough for this.

Fig. 9 A snapshot of the robot (shown distant) waiting on a human-operated car to exit a four-way stop intersection. This occurred at the day-to-day test site (see Section 4).

To determine intersection precedence at a four-way stop intersection, we first formed the position and orientation of simple polygon bounding boxes at the known given position of stop lines as indicated by the RNDF. Then, the estimated positions of vehicles are pulled from the dynamic obstacle map, as described by Section 3.2. When the robot stops at a stop line, it uses the presence of these vehicles in the

intersection-aligned bounding boxes to establish its own precedence at the intersection. To determine when it is its turn to proceed through the intersection, the robot detects the movement of cars through the intersection using the estimated position of *car* obstacles and a polygon representing the intersection's interior.

Intersections where the robot is required to come to a stop but cross traffic is not (*i.e.* a two-way stop) is particularly challenging. Because the on-coming traffic can be approaching at 30 mph, and the robot needs a 5 second gap (at least) to merge into traffic, the robot must be able to detect and distinguish oncoming cars at 67 meters. Adding to the problem, the robot must identify which approaching cars are truly coming to the intersection and which are simply on a neighboring, disjoint road segment. For those cars approaching the intersection, their time of arrival must be calculated. We do this by localizing the detected vehicles to the RNDF, and by estimating their forward velocity, estimate their time of arrival. So, for two-way stop intersections, the robot uses the position of vehicles stopped at the intersection at stop lines for precedence, while evaluating the time of arrival of the oncoming cars to which the robot must yield.

We use similar logic and the same perceptual elements to evaluate when the robot can change lanes on a multi-lane road. This lets the robot assess when it is safe to move between lanes that are moving in the same direction without cutting off another vehicle on the same road segment.

3.5 Route Planning

As stated in Section 1, the overall decision making process of the robot is decomposed into three subcomponents. A route planner assesses where the robot is now, and given the next goal location (from the DARPA Mission Definition File), plans a path over the RNDF to get to that goal. The RNDF is a simple weighted graph structure over which an implementation of Dijkstra's algorithm is sufficiently fast for mission-level path planning.

The RNDF is also used to encode temporary changes to the road network. For example, when a section of road is found to be blocked (as determined by lower-level control system), this information is lifted back up to the RNDF planner where a high cost is assigned to the edge in graph associated with the section of blocked road. This high cost is large enough to induce the robot to drive anywhere else over the RNDF to get to the goal, but if no other passage is found (or all other passages are found to also be blocked), the robot will try again to re-traverse a section previously found to be blocked.

3.6 Situational Awareness: Nested Hybrid Automata

A major difference between the DARPA Urban Challenge and the DARPA Grand Challenge was the additional complexity of the task in the urban environment. In addition to navigating through a static environment, the Urban Challenge required autonomous vehicles to interact with other moving vehicles (autonomous or otherwise), follow visually-marked lanes, and obey traffic laws. A major challenge in designing the software architecture for the Urban Challenge was providing the expressiveness necessary to capture the complexity of the task. Our approach to this challenge was to describe the task in a model based on a modified hybrid automaton.

A hybrid automaton is a model that captures both the continuous and the discrete aspects of a dynamical system. In particular, a continuous state (typically the position and velocities of the car) evolves concurrently with a discrete state (the current mode of operation), and we follow the standard definition of a hybrid automaton [**?**] as a tuple $H = (Q, X, E, U, f, G, R, x_0, q_0)$, where

- Q – the set of discrete states
- X – the continuous state space
- E – the set of events that can trigger transitions between different discrete states
- U – the input space
- $f : Q \times X \times U \rightarrow TX$ – encodes the evolution of the continuous state x as $\dot{x} = f(q, x, u)$
- $G : Q \times Q \times X \times (E \cup \varepsilon) \rightarrow \{0, 1\}$ – gives the guard conditions that triggers transitions between discrete states. In particular, a transition occurs between q to q' if the continuous state is x, the external event is $e \in E$ or possible the "empty event" ε (no event happened) if $G(q, q', x, e) = 1$
- $R : Q \times Q \times X \times E \rightarrow X$ – encodes the reset condition, in that the continuous state is reset to $R(q, q', x, e)$ when the system transitions from q to q' at continuous state x under event e
- $q_0 \in Q$ – initial discrete state
- $x_0 \in X$ – initial continuous state

Different definitions of such hybrid dynamics have been given, but they all share these basic building blocks in terms of continuous and discrete dynamics, guards, and resets.

An example of this is seen in Figure 10. In the figure, the discrete state starts out at q_0 and the continuous state evolves from x_0 according to $\dot{x} = f(q_0, x, u)$ until time τ. At that time, the continuous state is at $x(\tau_-)$ and event e happens. The guard condition $G(q_0, q', x(\tau_-), e)$ becomes 1 and the discrete state transitions from q_0 to q'. The continuous state is reset to $x(\tau_+) = R(q_0, q', x(\tau_-), e)$, from which it evolves as $\dot{x} = f(q', x, u)$.

Certainly, the task of completing the Urban Challenge could be expressed as a single, very large hybrid automaton. One would start by enumerating all the necessary control modes for the vehicle, and then define the connectivity between the control modes based on perceptual signals. However, for a sufficiently complex task, this representation quickly becomes cumbersome to work with.

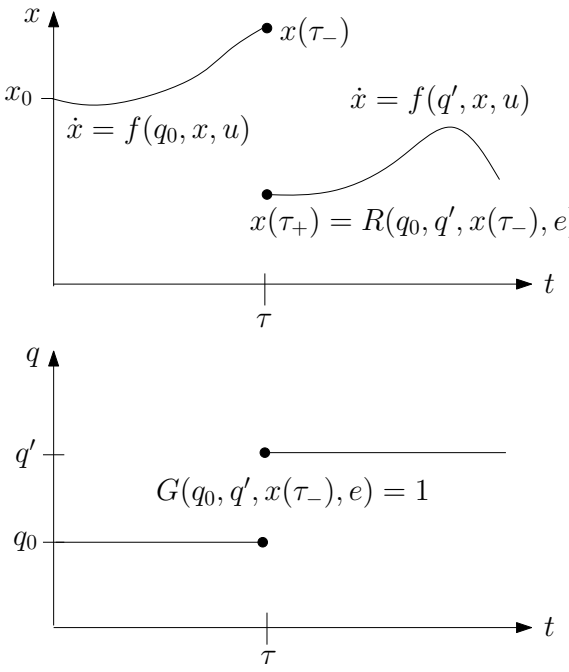


Fig. 10 Evolution of a hybrid automaton.

While the distributed structure of the hybrid automata keeps execution by the control system tractable (since the control system needs only execute the current control mode and check the current mode's out-going guards, which increase linearly with the number of modes), management of the system by developers can quickly become difficult. Our solution to this issue is the development of a nested hybrid automaton, which makes use of the natural structure of the task and environment to decrease the complexity by grouping related control modes into a hierarchical structure.

A nested hybrid automaton differs from a hybrid automaton in that it is defined recursively. Just as in a hybrid automaton, a finite automaton maintains the discrete state of the system. These discrete states can be interpreted as modes of operation for the robot, mapping to a function relating continuous state to continuous control. However, in a nested hybrid automaton, each state in the finite automaton represents either a discrete state (as in the hybrid automaton) or another sub-automata which maintains a a more fine-grained representation of the discrete state in a hierarchical manner. The advantages over a standard hybrid automaton is an increased compactness of representation, modularity and isolation, as well as the ability to express discrete state transitions at different levels of abstraction asynchronously.

As an example, take the highest-level states used to define the operation of the Sting vehicle:

- `operator-run` - navigate through the urban environment.
- `operator-pause` - stop the vehicle and wait for operator input.
- `operator-stand-by` - sound the warning siren before transitioning into `operator-run`.

The only non-trivial of these states is `operator-run`, which corresponds to the operator/user putting the vehicle in an autonomous run-mode. The `operator-run` state is itself expanded into an automata that describes the modes of operation and transitions between these modes.

As seen in Figure 11 the hybrid automaton that corresponds to the `operator-run` mode has the discrete states

- `follow-lanes` - navigate along marked roads.
- `handle-intersection` - navigate through intersections of marked roads.
- `overtake-static-obstacle` - navigate around an obstacle in the road by driving in the oncoming lane.
- `execute-u-turn` - turn around on a marked road because of an obstacle completely blocking the road.
- `park` - navigate into a marked parking spot.
- `unpark` - navigate out of a marked parking spot.

Each of these modes in turn contain their own hybrid automata. The `follow-lanes` mode is a hybrid automaton whose discrete states define the modes of operation needed to navigate along marked roads:

- `follow-lane` - keep the vehicle between the perceived lane markings of the planned road.
- `blind` - navigate along the road using GPS and LIDAR as the primary sensors. (The vehicle transitions to this state when the vision sensor has failed, e.g. because the vehicle is facing the sun.)
- `blocked` - come to a stop because of an obstacle in the planned lane.
- `should-overtake` - get ready to overtake an obstacle blocking the planned lane.

Each of these modes is a primitive discrete state in that they map directly to a mode of operation and do not expand further to into automata. Figure 11 illustrates this portion of the nested finite state automata used to describe the process of urban driving. In this figure, the automata defined within the `operator-run` state is depicted on the left. Within the `follow-lanes` state is nested an automata that steps through the process of driving in a lane based on perceptual signals, including the states `follow-lane`, `blind`, `blocked` and `should-overtake`, shown on the right.

Using the recursive structure of the nested hybrid automata, we were able to represent and manage the complexity of the tasks of the Urban Challenge. High-level tasks were represented as states in the highest level of the recursive definition. Subtasks were represented at lowers levels within each high-level task. This provided a measure of isolation for the definition of these sub-tasks, as transitions between

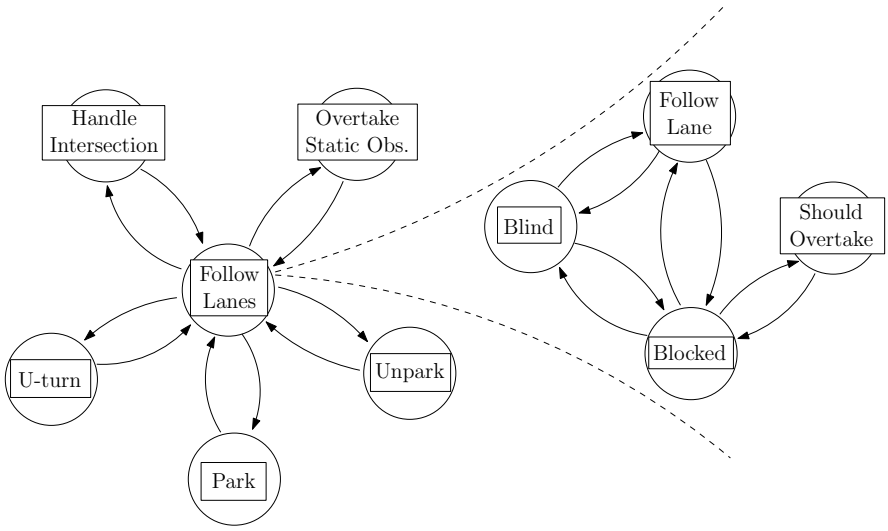


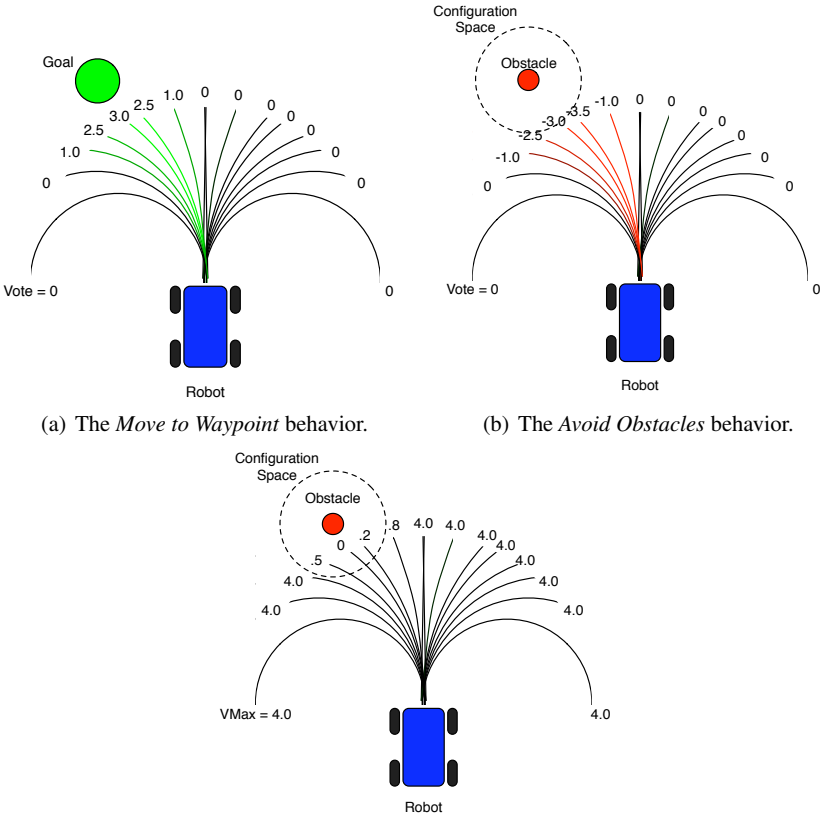
Fig. 11 A portion of the nested finite state automata used to describe urban driving. The automata defined within the `operator-run` state is depicted on the left. Within the `follow-lanes` state is nested an automata that steps through the process of driving along marked roads, shown on the right. This automata is made up of four modes of operation: driving in a lane based on perceptual signals (`follow lane`), falling back on LIDAR/map-based navigation when perceptual signals fail (`blind`), pausing behind another vehicle stopped in the lane (`blocked`) and overtaking said vehicle (`should overtake`).

high-level tasks could be defined independently of the state of the sub-tasks below them.

3.7 Steering and Speed Control

Steering and speed control is implemented in a behavior-based voting design [**?**, **?**]. In this design, a number of behaviors evaluate candidate actions over a short temporal scale, each behavior representing a specific interest pertaining to the robot's objective.

In this implementation, as shown in Figure 12, the behaviors reason over constant curvature arcs. Each behavior distributes an allocation of votes over an array of potential arcs for the robot to navigate along. The behaviors can allocate votes for arcs that work to achieve its interests, or against arcs that are detrimental to its interests. In addition to distributing votes for or against arcs, behaviors assign a maximum allowable velocity, associated with each arc. Behaviors need not necessarily express an interest across both curvature and velocity. A behavior may vote for curvatures and leave the allowable velocities set to the robot's maximum veloc-



(a) The *Move to Waypoint* behavior.

(b) The *Avoid Obstacles* behavior.

(c) The *Maintain Headway* behavior.

Fig. 12 Three of the behaviors used in the implementation of the reactive layer. In (a), the *Move to Waypoint* behavior votes in support of curvatures that take the robot closer to the provided waypoint. In (b), the *Avoid Obstacles* behavior votes against curvatures that take the robot toward sensed obstacles. In (c), the *Maintain Headway* behavior sets a maximum allowable translational velocity for each curvature, with respect to sensed obstacles. An arbiter tallies the weighted votes provided by the set of behaviors, and outputs the curvature with the most votes, and the minimum allowable translational velocity for that curvature.

ity, it may cast no votes for or against curvatures and express its interest across the allowable velocities, or it may express its interest across both dimensions.

To choose a curvature and velocity for the robot to execute, an arbiter sums the votes cast by each behavior for each curvature arc, weighting the votes for each behavior according to a predetermined weighting scheme. It selects for execution the curvature arc with the highest total of votes. It then selects for execution the minimum of the maximum allowable velocities assigned by the respective behaviors to the selected curvature arc. The selected curvature and velocity are sent on to low-level controllers for execution.

Over 30 different behaviors were used in combination to create 25 different control modes. Three of the behaviors are diagrammed in Figure 12:

- *Move to Waypoint* - shown in Figure 12(a), allocates positive votes to arcs according to a linear control law relating the local heading to the waypoint to a commanded curvature.
- *Avoid Obstacles* - shown in Figure 12(b), allocates negative votes to arcs according to the distance along the arc that the arc crosses into the configuration space around a detected obstacle. Arcs that do not cross into the configuration space of the obstacle are not voted against.
- *Maintain Headway* - shown in Figure 12(c), sets maximum allowable velocities for each arc according to the distance along the arc that the arc crosses into the configuration space around a detected obstacle. If the arc does not cross into the configuration space of the obstacle, the robot's maximum speed is assigned. If the arc crosses into the configuration space of the obstacle within a parameterized safety distance, the maximum allowable velocity is zero.

4 Vehicle Performance

In this section, we describe how the system we built performed in our own field trials, during the DARPA site visit, and during the DARPA National Qualifying Event. In general, we found that our system performed well against various challenging scenarios: high speed driving, irregular intersections, road navigation with GPS denial, multi-lane road navigation, U-turns on irregular road segments with moving obstacles.

Testing occurred at a handful of locations. Day-to-day testing of primary tasks took place at our on-site facility shown in Figure 13. Here, we developed algorithms for speed and steering control at up to 25mph, lane keeping, obstacle field navigation, parking, overtaking road blockages, headway maintenance, and intersection handling. This was also the location used for our DARPA site visit.

Fig. 13 A panoramic photograph of the day-to-day testing site used by Sting Racing.

Testing of high speed steering and speed control, headway maintenance, lane changing, and high speed obstacle avoidance occurred at the high-speed driver training track at the Georgia Public Safety Training Center (GPSTC) in Forsyth, Georgia. GPSTC is a state-run facility for (among other things) training police officers in high performance, high speed driving both on long road segments, as well as among close-quarters intersections. This track is shown in Figure 14. This course was particularly challenging because it was specifically designed to be difficult to handle at high speeds. For example, the roadway has several rolling hills with extraordinarily steep slopes. The road curves counter-intuitively just after the crests of several large hills. In all, this course was ideal for stress testing the robot's ability to handle driving scenarios at higher speeds.

Fig. 14 A satellite image of the Georgia Public Safety Training Center's 1.5 mile high speed test track.

Figure 15 illustrates an example of a test we performed on our robot at the GPSTC high speed track. Figure 15(b) shows the robot maintaining headway behind another vehicle (in this case, at about 30 mph). As shown in Figure 15(c), in a quick motion, the lead vehicle changes lanes, revealing a previously occluded stationary obstacle. If the robot quickly and accurately detects the barrels, the robot should be able to stop prior to collision. (In this case, the left lane may contain oncoming traffic, so the robot must come to a complete stop before changing lanes.) Figure 15(d) shows that the vehicle successfully stopped safely. Subsequently, the robot backed up slightly, and performed an overtake maneuver, and continued on its way.

Also at the GPSTC, we tested on a course specially designed to stress a driver's ability to safely navigate intersections (see Figure 16). On this course, we developed and evaluated our vehicle's ability to negotiate intersections, with special attention to badly-behaving drivers and pulling out from four-way stop and two-way stop intersections.

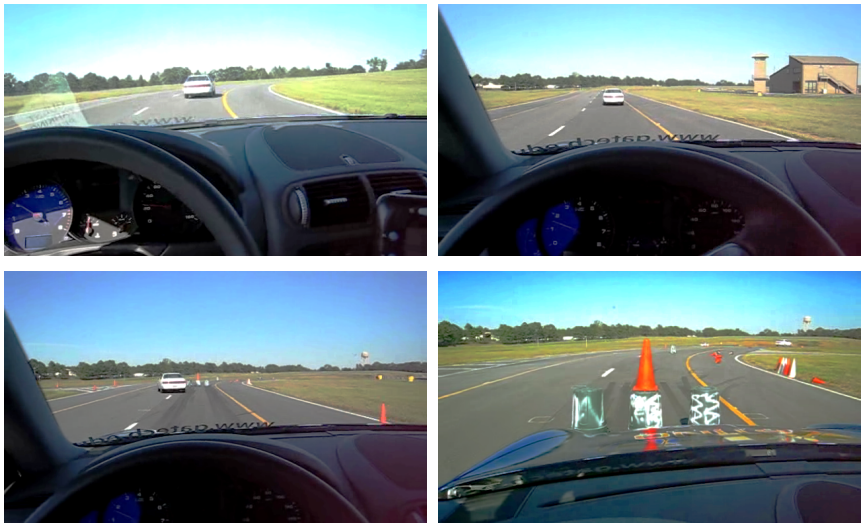


Fig. 15 (a) The robot driving at 25+ mph on the GPSTC high speed track. Taken from safety operator's position. (b) The robot shown following another vehicle at speed. (c) The lead vehicle darts away at the last moment to suddenly expose an occluded stationary object. (d) The robot stops in time before the obstacle.

Fig. 16 A satellite image of the GPSTC urban course.

In addition to standard driving and intersection handling, the GPSTC urban course allowed us to test other important test scenarios. Several of the intersections are formed from irregularly angled roads. To be successful in this type of scenario, the robot must be robust in its analysis of the presence of other vehicles at an intersection (as discussed in Section 3.4.) Figure 17 depicts the robot queuing at an intersection behind another vehicle.

Fig. 17 The robot shown queuing behind another vehicle at an intersection on the GPSTC urban course.

4.1 National Qualifying Event

The National Qualifying Event was comprised of three courses. Here, we describe the performance of our vehicle at these three courses, as well as draw some conclusions about what could have improved the performance of our vehicle.

4.1.1 Course C

Course C at the National Qualifying Event was designed to test a robot's ability to navigate four-way stop intersections, identify road blockages, and execute a U-turn. Our robot was able to successfully navigate the outer loop, passing through both intersections. When it approached the road blockage on the southern loop, it properly identified it as such, performed a u-turn on the road, and resumed its mission on another route to the active goal point.

4.1.2 Course A

Course A at the National Qualifying Event was constructed to test a robot's ability to handle pulling out from a stop at a two-way intersection. That is, the robot comes to a stop sign at an intersection where traffic on the cross street does not have to stop. We placed our robot at the starting position, initiated its various processes and quickly verified that all systems had kicked off properly. From the point of view of a spectator, when the robot was enabled by the wireless control system, it approached the intersection, slowed momentarily, and collided with the wall on the far side of the intersection. The collision was precipitated by the simultaneous occurrence of two failures.

First, the serial connection between the pose estimation computer and the GPS device hung in a way we did not anticipate. The connection appeared to be alive,

but the position information it provided was "stuck" at a fixed point in time. As the car decided to approach the intersection, it increased the input to the accelerator. In reality, it was moving towards the intersection, but because of the error on the GPS serial connection, the robot believed itself to be stationary.

Second, because the robot believed itself to be stationary, the robot calculated that the wall (detected via the LIDAR scanners) had an increasing and positive velocity. Ground returns – points on the ground detected by the LIDAR scanners – had been a common occurrence during our field testing where the ground was hilly. And because walls have zero velocity and ground returns very often have a high velocity, we had programmed the robot to estimate that what it was seeing was a ground return.

As the other cars on the course approached the robot, their predicted trajectories crossed that of the robot, and the robot attempted to stop as a defensive measure. Unfortunately, this occurred too late for the vehicle to stop in time.

Over many months of testing, we had seen the serial connection to the GPS sensor become livelocked only once before. At the time, we took steps we believed to correct the problem in hardware. As a result of the collision, the front sensor mount was bent, but none of the sensors themselves were damaged at all. The sensor mount was rewelded to the front of the vehicle and the sensors reattached and recalibrated. The next day, we were pleased to see our robot out in testing on the course.

4.1.3 Course B

Course B at the National Qualifying Event was the largest course, and required the robot exit the starting dock, proceed around the large traffic circle, navigate the road network, and park and unpark from a spot in the parking area.

Our robot successfully navigated onto the traffic circle and made the turn onto the narrow two-lane roadway known as the "chute" that leads out to the larger section of the course. We decided to end the run while the robot was still in the chute because of the apparently confused behavior the robot exhibited.

The chute was constructed by placing K-rails right on top of the border of the lanes. This made the road very narrow. We designed the robot to give strict regard to the rules governing vehicle-environment spacing. Specifically, 1 meter separation on both sides of the vehicle had to be maintained at all times, while not crossing a double-yellow line. The K-rails in the chute were so close to the road as to necessitate the violation of this rule.

When presented with this situation, the robot entered the "Overtake Obstacle" mode, which allows it to cross a double-yellow line in order to get around a blockage. But because the K-rail on the opposite side of the road was also so close, the robot was not able to make progress while in overtake mode. In a sense, the robot became "wedged" in the chute. What we failed to anticipate after months and months of making sure that the robot could follow the rules successfully, was that sometimes rules need to be bent a little. Any human driver would easily have identified

the chute-setup as a non-regular situation, and would have improvised his way of out the predicament. This, alas, was something our system was ill-equipped to do.

5 Conclusion

Overall, we are happy with many elements of our vehicle's performance. Our system was able to do a number of things effectively, some of which were not tested at the National Qualifying Event. This includes our system's ability to detect and track dynamic obstacles based on the fusion of horizontal planar LIDAR scanners and RADAR scanners. Our robot's ability to drive at full speed on long, hilly stretches among other vehicles is an accomplishment. The nested hybrid automata approach made development and testing of the complex problem of situational awareness achievable for our small team of researchers.

6 Acknowledgements

We thank the various groups at Georgia Tech who funded most of this work, including the College of Computing, the school of Electrical and Computer engineering, the Georgia Tech Research Institute, and the office of the Vice Provost. Thanks to the Georgia Public Safety Training for use of their facilities for testing our vehicle. Thanks to everyone who contributed to the project. We had very generous support from SAIC in terms of staff and equipment. We would like to thank in particular Robert Schafrik and Karl Kluge from SAIC. In addition, a number of graduate and undergraduate students at Georgia Tech contributed to the project with ideas, early implementations and stress testing of the project.